VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Autonomní řízení modelu auta pomocí Free RTOS

Autonomous Control of a Car Model Using Free RTOS

2019 David Štěpaník

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **David Štěpaník**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Autonomní řízení modelu auta pomocí Free RTOS
Autonomous Control of a Car Model Using Free RTOS

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je analyzovat stávající program pro autonomní řízení modelu auta po vyznačené dráze a následně provést převod tohoto programu do Free RTOS. Důležitým krokem pro úspěšnou implementaci je správné rozdělení programu na samostatné úlohy a jejich následnou synchronizaci. Pro realizaci využijte Kit FRDM-K64F.
1. Seznamte se se základní koncepcí systému FreeRTOS. Seznamte se s programem pro autonomní řízení modelu auta.
2. Připravte několik příkladů návaznosti technického vybavení mikropočítače (časovače a sériové linky) na fungování procesů ve FreeRTOS.
3. Navrhněte rozdělení a organizaci procesů pro řízení modelu auta pro FreeRTOS.
4. Navrženou koncepci otestujte a vyhodnoťte funkcionalitu a spolehlivost.
5. Doplňte řídící program o záznam a průběžné odesílání dat přes Ethernet.
6. Doplňte řídící program o příjem řídících a konfiguračních dat.
7. Vyhodnoťte celkové fungování aplikace, výkonovou rezervu procesoru a plánovatelnost procesů.

Seznam doporučené odborné literatury:

[1] Free RTOS http://www.freertos.org
[2] Using the FreeRTOS Real Time Kernel, http://www.freertos.org
[3] Kinetis design studio http://www.nxp.com
[4] Kinetis SDK Builder, http://www.nxp.com

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Olivka, Ph.D.**

Datum zadání: 01.09.2017
Datum odevzdání: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2019 .....

Rád bych poděkoval svému vedoucímu práce Ing. Petru Olivkovi, Ph.D. za odbornou pomoc a trpělivost při tvorbě této práce.

Abstrakt

Tato bakalářská práce popisuje převod stávajícího programu pro autonomní řízení modelu auta do programu, využívajícího FreeRTOS. Program je rozdělen do úloh, které jsou sesynchronizovány s časováním stávajícího programu. Do programu je přidáno textové uživatelské rozhraní a možnost odesílat data přez ethernetový port. Navržená koncepce je naimplementována k použití na desce FRDM-K64F. Výsledný systém je otestován a vyhodnocen.

Klíčová slova: Autonomní řízení, FreeRTOS, textové uživatelské rozhraní, ethernetový port, FRDM-K64F

Abstract

This bachelor thesis describes the conversion of existing program for autonomous self-driving car model into a program, using FreeRTOS. Program is divided into tasks, which are synchronized with timing of the existing program. A text user interface and an option to send data over Ethernet port are added to the program. The proposed concept is implemented for use on the FRDM-K64F board. The resulting system is tested and evaluated.

Key Words: Autonomous control, FreeRTOS, text user interface, ethernet port, FRDM-K64F

Obsah

Seznam použitých zkratek a symbolů

ADC	–	Analog-Digital Converter
API	–	Application Programming Interface
DIP	–	Dual Inline Package
FIFO	–	First In First Out
FTM	–	FlexTimer Module
GPIO	–	General Purpose Input Output
LED	–	Light Emitting Diode
NiMH	–	Nickel Metal Hydride
PWM	–	Pulse Width Modulation
RAM	–	Random Access Memory
RC	–	Radio Control
RGB	–	Red Green Blue
RTOS	–	Real Time Operational System
SDK	–	Software Development Kit
UART	–	Universal Asynchronous Receiver Transmitter

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Cílem této bakalářské práce je analyzovat stávající program pro autonomní řížení modelu auta po vyznačené dráze a následně převést tento program do FreeRTOS.

Tato práce vychází z několika bakalářských prací, které zpracovávají obraz z řádkové kamery [1], jíž je osazen model automobilu, počítají optimální dráhu vozu[2] a umožňují zobrazit textové uživatelské rozhraní pomocí sériové linky[3].

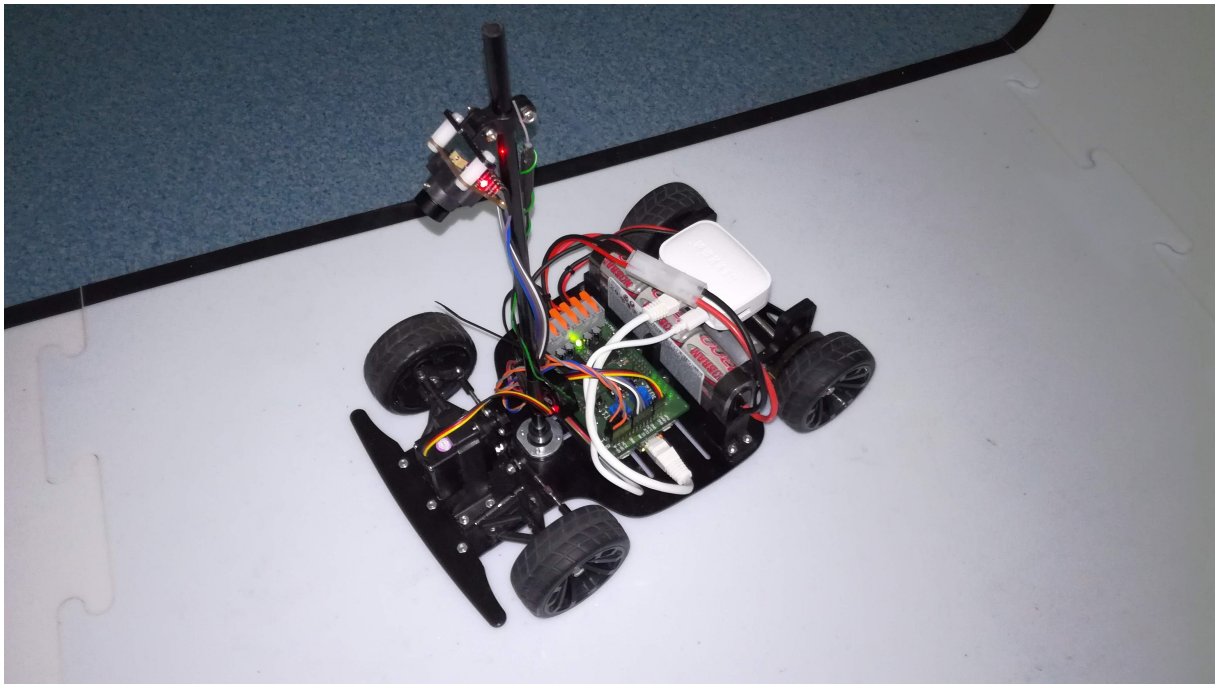
Autonomní řízení vozu je v dnešní době velice aktuální téma, které má však spoustu odpůrců. Ovládání vozu s živými pasažéry vyžaduje maximální preciznost a klade důraz na bezpečnost pasažérů. Doposud vyvinuté systémy nejsou dokonalé a tedy nemohou zajistit stoprocentní spolehlivost provozu za jakýchkoli podmínek. Z tohoto důvodu je autonomní řízení automobilů vnímáno spíše skepticky, avšak vývoj těchto systémů jde velmi rychle kupředu a nebude dlouho trvat, než se podaří vyvinout systém schopný komerčního provozu.

V této práci bude navržen program, vycházející ze stávajícího programu autonomního řízení modelu auta, využívající FreeRTOS k synchronizaci a plánování úloh. Celý program bude rozdělen do jednotlivých úloh, jimž bude přiřazena priorita. Do programu bude přidáno textové uživatelské rozhraní a možnost odesílat data přez ethernetový port pomocí UDP protokolu. Program bude implementován k použití na desce FRDM-K64F s použitím MCUXpresso SDK API[5].

Tímto způsobem bude zajištěn determinismus programu a synchronizace úloh. Program tak bude efektivnější a plánovatelnější, než byl doposud.

2 Model Automobilu

K uskutečnění této bakalářské práce byl využit model automobilu zvaný Alamak vyroben společností Landzo[13]. Pohon modelu je zajištěn dvěma stejnosměrnými motory a řízení jedním servomotorem, který natáčí přední kola modelu. K řízení systému je model vybaven deskou FRDM-K64F, nebo deskou FRDM-K66F a modulem POLI-TFC, který slouží k propojení všech komponentů. Ze středu modelu je vyvedena tyč, na které je umístěna řádková kamera, snímající vždy 128 pixelů v jednom řádku. Na stejnou tyč lze umístit rádiový přijímač, pro RC řízení modelu. Celý systém je napájen 7.2V NiMH baterií. Pro bezdrátovou komunikaci je využit router v režimu přístupového bodu, který je k desce připojen ethernetovým kabelem. Sestavený model se všemi periferiemi, vyjímaje rádiového přijímače je vyobrazen na obrázku 1.



Obrázek 1: Model auta Alamak

3 Řídící moduly

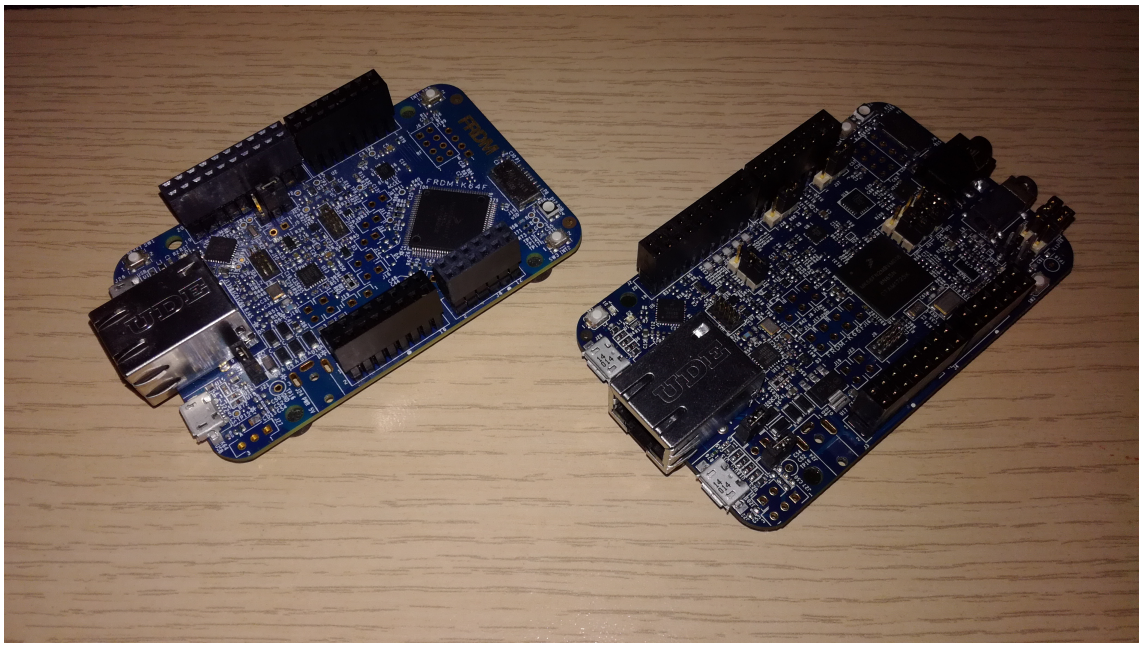
Řídícími moduly se rozumí desky, použité pro řízení a propojení jednotlivých komponentů systému. V tomto projektu jsou použity desky FRDM-K64F, FRDM-K66F a POLI-TFC.

3.1 Deska FRDM-K64F

Deska od společnosti NXP Semiconductors je osazena mikroporcesorem s jádrem ARM Cortex-M4 s taktem 120 MHz, 1 MB flash paměti a 256 KB RAM paměti. Deska je dále osazena čipem pro OpenSDA, jenž slouží jako ladící a sériový adapter a také nahrává programy do paměti mikroprocesoru. Slouží tedy společně s USB portem k propojení s počítačem, či jiným zařízením. Deska je nadále osazena RGB LED diodou, ethernetovým portem a GPIO piny pro připojení dalších komponentů. Mezi další periferie této desky patří například slot pro microSD kartu, akcelerometr a magnetometr, nebo dvě tlačítka, avšak tyto součásti nejsou v této práci využity.[9]

3.2 Deska FRDM-K66F

Deska je velice podobná FRDM-K64F, liší se taktem jádra 180 MHz a velikostí Flash paměti, která je 2 MB. Oproti FRDM-K64F má FRDM-K66F navíc také audio kondek, vstup pro analogový mikrofon, výstup pro sluchátka/analog, digitální mikrofon a také gyroskop. Rozdílný je také pinout, kvůli kterému jsou k této práci použity dvě verze desky POLI-TFC. [10]



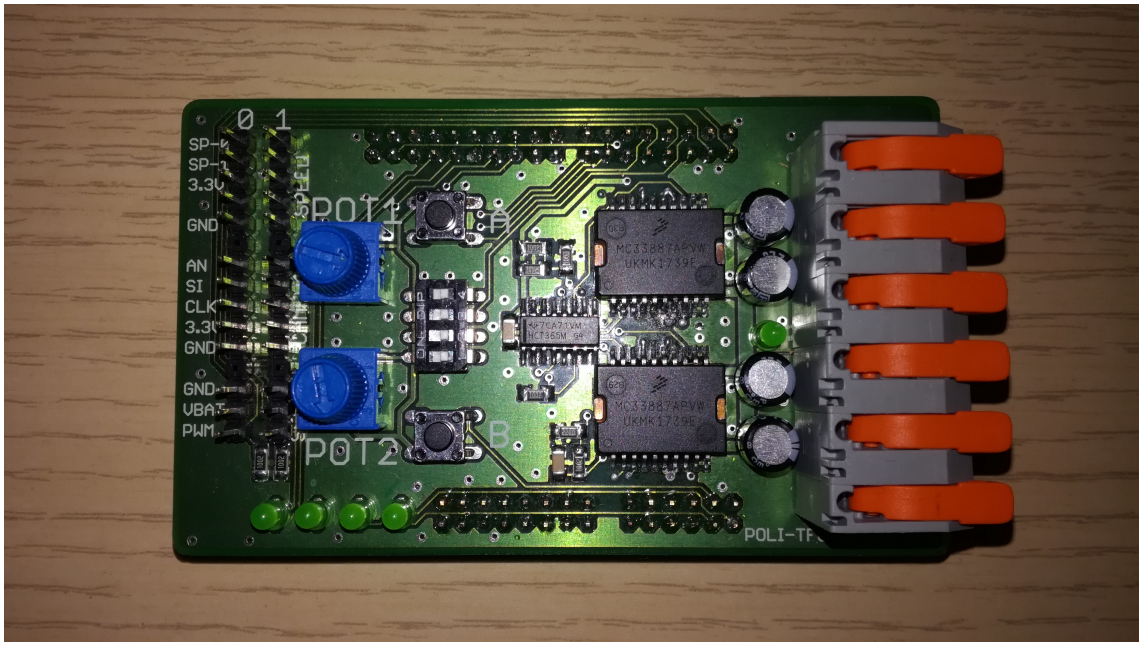
Obrázek 2: FRDM-K64F a FRDM-K66F

3.3 Modul POLI-TFC

Jedná se o klon desky FRDM-TFC[11], která byla společností NXP Semiconductors navržena pro účely soutěže NXP Cup[12].

Deska je osazena dvěma H-mosty pro ovládání motorů pomocí PWM, čtyřmi LED diodami, které lze využít například pro indikaci stavu baterie a periferiemi pro uživatelský vstup. Mezi uživatelské vstupy patří například dvě tlačítka, dva potenciometry a čtveřice DIP přepínačů. Do tohoto modulu se rovněž zapojuje 7.2V NiMH baterie, která napájí celý systém.

Vzhledem k odlišnému uspořádání pinů na deskách FRDM-K64F a FRDM-K66F je nutné využití dvou různých desek, ty se však liší pouze v rozložení pinů, funkcionalita je totožná.



Obrázek 3: Modul POLI-TFC

4 FreeRTOS[6]

V této kapitole je popsán operační systém FreeRTOS a jeho hlavní části. Mezi tyto části patří například úlohy, které tvoří vlákna programu řízená plánovačem, meziúlohová komunikace ve formě front zpráv a synchronizace pomocí semaforů. FreeRTOS nabízí vývojářům FreeRTOS API[7], které je v této práci použito.

4.1 Operační systém reálného času

Operační systém je programové vybavení zajišťující souběžnost několika úloh na jednom procesoru tak, aby to vypadalo, jako by běžely najednou. Kdybychom však zpomalili za sebou navazující instrukce procesoru, zjistili bychom, že to reálně není možné. Procesor, respektive každé jeho jádro je schopno zpracovávat současně pouze jedno vlákno. Operační systém tedy obsahuje plánovač, který rozhoduje, kdy je který proces zpracováván a vytváří tedy iluzi paralelního běhu programů velice rychlým přepínáním mezi jednotlivými programy.

Operační systémy se liší tím, jak plánovač rozhoduje, kdy spustit kterou úlohu. Například plánovač některých serverů Unix se pokusí vždy rozdělit výpočetní čas mezi uživatele rovným dílem. Naopak plánovač stolních operačních systémů, jako třeba Windows, se pokusí zajistit plynulý provoz právě užívané aplikace a zůstat responsivní.

Plánovač operačních systémů pracujících v reálném čase klade důraz na časové požadavky, což znamená, že systém musí odpovědět na danou událost v daném časovém termínu. Tohoto je plánovač schopen dosáhnout pouze tehdy, je-li deterministický, tedy předvídatelný.

Běžné plánovače pracující v reálném čase, jako například ten, který je použit ve FreeRTOS, zajišťují determinismus tím, že dovolí uživateli přiřadit vláknům priority. Dle přiřazených priorit je pak plánovač schopen rozhodovat mezi jednotlivými vlákny.

4.2 FreeRTOS

FreeRTOS je třídou RTOS, která je dost malá na to, aby mohla být použita v mikrokontrolérech, což ji však nijak neomezuje pouze na tyto aplikace.

Mikrokontroléry jsou počítače s omezenými zdroji. V jediném čipu lze nalézt výpočetní procesor, ROM nebo Flash paměť pro ukládání kódu programu a také RAM, potřebnou pro spuštěné programy.

Mikrokontroléry se používají ve vestavěných zařízeních, která splňují většinou velice konkrétní úkoly. Málokdy je RTOS těmito úkoly, vzhledem k jejich určení, plně využit. FreeRTOS z toho důvodu zajišťuje pouze plánování v reálném čase, mezi vláknovou komunikaci, časování a synchronizační prvky. Jestliže aplikace vyžaduje další funkcionalitu, jako například textové uživatelské rozhraní v příkazové konzoli nebo síťovou komunikaci, je možné tyto komponenty přidat.

4.3 Úlohy

Úlohy bychom na běžném operačním systému nazvali vlákna. Jedná se tedy o vlákna programu, která jsou plánovačem spouštěna a pozastavována na základě akcí v reálném čase. Úlohy mohou existovat ve čtyřech různých stavech.

4.3.1 Stav úlohy

1. *Spuštěná úloha*

   Jestliže úloha právě probíhá a obsazuje tedy kontext procesoru, říká se jí *spuštěná úloha*. Na jednojádrovém procesoru může být spouštěna pouze jedna úloha v daném čase.

2. *Připravená úloha*

   Připravená úloha je ta, která může být spuštěna, ale momentálně není, protože jiná úloha se stejnou nebo vyšší prioritou je právě *spuštěná*.

3. *Blokovaná úloha*

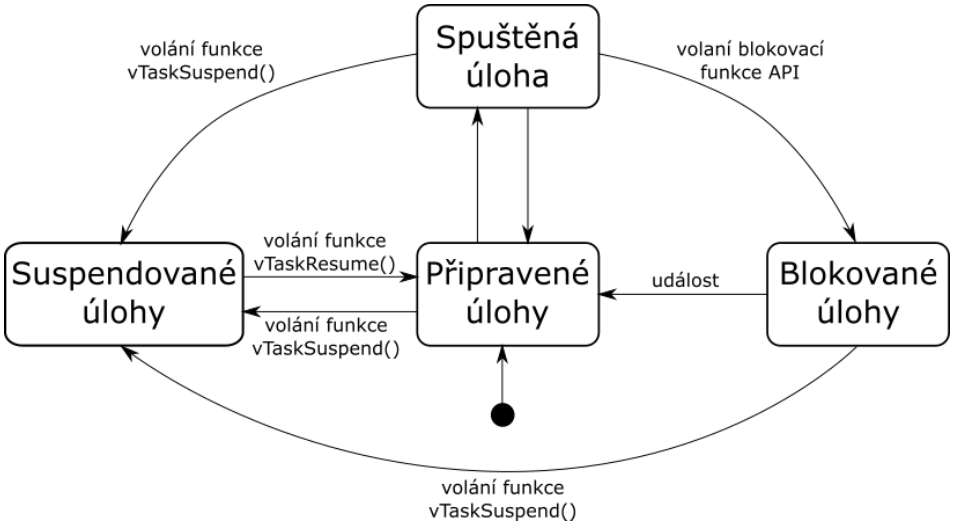
   Úloha je *blokovaná* jestliže právě čeká na nějakou dočasnou, nebo externí událost. Například když úloha zavolá funkci `vTaskDelay()`, bude dočasně blokována po dobu specifikovanou parametrem funkce. Úlohy ale mohou být také blokovány při čekání na prvky ve frontě, semafor nebo notifikaci. Běžně mají úlohy v *blokovaném* stavu také timeout periodu, po jejichž uplynutí dojde k odblokování úlohy a úloha se tak dostane do stavu *připravena*.

   Blokované úlohy nikdy nemohou být spuštěny a tedy nikdy nebudou zabírat výpočetní čas.

4. *Suspednovaná úloha*

   Stejně jako *blokované úlohy*, ani *suspendované úlohy* nemohou být spuštěny, avšak *suspendované úlohy* nemají timeout periodu, po které by se dostaly do stavu *připravena*. Do stavu *suspendovaná* se tedy mohou úlohy dostat pouze voláním API funkce `vTaskSuspend()` a zpět do stavu připravena voláním funkce `vTaskResume()`.

Všechny možné změny stavu jsou uvedeny ve stavovém diagramu na obrázku 4.



Obrázek 4: Stavový diagram možných změn stavů úloh

4.3.2 Priorita

Při inicializaci úlohy je každé úloze přiřazena priorita, která může mít hodnotu od 0 až po (`configMAX_PRIORITIES`−1), kde `configMAX_PRIORITIES` je definovatelné v hlavičkovém souboru `FreeRTOSConfig.h`.

Tuto hodnotu lze nastavit na libovolné číslo, avšak v rámci úspory paměti RAM, by měla být nastavena na co nejnižší potřebnou hodnotu.

Úlohy s nízkou hodnotou priority mají rovněž nízkou prioritu. Ačkoli se tato informace jeví jako logická a intuitivní, tak ne vždy tomu tak musí být. Například priority přiřazené přerušením na použitých procesorech mají shodné číslování, leč opačnou váhu priorit. Přerušení s prioritou 0 má tedy vyšší prioritu, než přerušení s prioritou 1. Abychom mohli využít API funkcí přímo v přerušení, je nutné, aby tato přerušení měla prioritu nižší, než přerušení kernelu FreeRTOS, aby mohl kernel na tato volání reagovat. Žádná hodnota priority přiřazena úloze nebude mít vyšší prioritu, než přerušení s nejnižší prioritou.

Plánovač vždy zajistí, že ve stavu *spuštěná* bude právě ta úloha, která má nejvyšší prioritu ze všech *připravených* úloh. Libovolný počet úloh může mít stejnou nejvyšší prioritu, v tom případě bude spuštěna ta úloha, která čeká na spuštění nejdéle. K tomu dojte až poté, co se předchozí *spuštěná* úloha dostane do stavu *blokovaná*, nebo *suspendovaná* a nebo sama umožní jiné úloze přejít do stavu *spuštěná* zvoláním API funkce `vTaskYIELD()`. V případě, že bude `configUSE_TIME_SLICING` nastaveno na 1, pak úlohy se stejnou nejvyšší prioritou budou rovnoměrně sdílet výpočetní čas.

4.3.3 Idle úloha

Tato úloha je automaticky vytvořena při spuštění plánovače, aby bylo zajištěno, že existuje vždy alespoň jedna úloha, která může být *spuštěná*. Při vytvoření je jí přidělena nejnižší možná priorita a to 0, aby bylo zajištěno, že nikdy nebude brát výpočetní čas úlohám s vyšší prioritou.

Idle úloha je rovněž zodpovědná za uvolnění paměti po úlohách, které byly vymazány pomocí funkce `vTaskDelete()`. U aplikací, ve kterých dochází k mazání úloh je tedy nutné zajistit, aby Idle úloha dostala možnost paměť po smazaných úlohách vyčistit.

V případě, že existuje úloha se stejnou prioritou, jako Idle úloha, je potřeba, aby bylo `configIDLE_SHOULD_YIELD` nastaveno na 1. V opačném případě může dojít k situaci, kdy se úloha vůbec nedostane do stavu *spuštěná*, protože Idle úloha bude neustále probíhat.

4.4 Meziúlohová komunikace a synchronizace

K úspěšné synchronizaci a komunikaci mezi úlohami je zapotřebí fronty zpráv. Tato fronta je ve FreeRTOS využita pro různé případy, ať již jako semafor, nebo jako prostředek pro sdílení dat mezi úlohami.

4.4.1 Fronta zpráv

Fronta je hlavní způsob meziúlohové komunikace a synchronizace. Mohou být použity k posílání zpráv či dat mezi úlohami, nebo mezi přerušeními a úlohami. Fronta je také bezpečná pro vícevláknové aplikace, neboť při inicializaci dojde k alokování paměti pro celou frontu, tedy velikost jednoho prvku vynásobena maximálním počtem prvků ve frontě. Když pak úloha, nebo přerušení zašle data do fronty, data se uloží do této paměti celá, nikoli jen ukazatel na data.

Do fronty lze zapisovat od konce, ale i na začátek. Nejběžnější využití fronty je FIFO, kdy prvek vložen do fronty jako první, je také jako první vyjmut.

Díky frontě může úloha přejít do stavu blokována a to ať už vložením do plné fronty, nebo při pokusu vyjmout prvek z prázdné fronty. V takovém případě je úloha blokována, dokud se fronta neuvolní, nebo nezaplní, popřípadě nedojde k vypršení časového limitu.

V případě, že je více úloh blokováno jednou frontou, bude odblokována ta, která má vyšší priortu, popřípadě ta, která byla blokována jako první.

4.4.2 Semafory

K synchronizaci vytvořených úloh můžeme použít semafory. Semafor si lze předsavit jako frontu s jedním, či více prvky, který má jistá pravidla pro vyjmutí a vložení prvku. Abychom lépe pochopili význam semaforů je třeba si vysvětlit pojem kritická sekce a vzájemné vyloučení.

V případě paralelního, nebo pseudoparalelního běhu aplikace může docházet k přístupu do sdílené paměti několika vlákny současně. V takovém případě by mohlo dojít k přepsání dat dvěmi, či více vlákny najednou, což by vedlo ke špatným výsledkům. Část kódu, kde přistupujeme

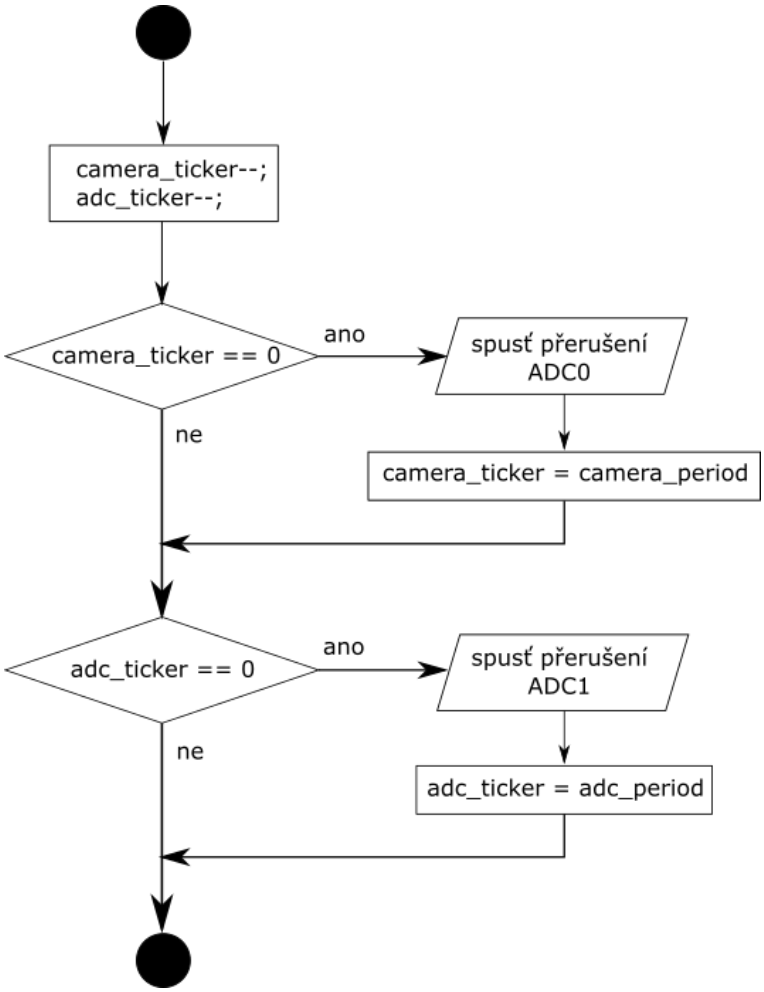
k těmto proměnným, se nazývá kritická sekce a je nutné zajistit, aby k datům přistupovalo vždy pouze jedno vlákno a bylo tedy zajištěno vzájemné vyloučení.

Pro tyto účely FreeRTOS nabízí k dispozici binární semafory, čítající semafory, mutexy a rekurzivní mutexy.

Abychom mohli synchronizovat přerušení a úlohy, je nutné používat funkce bezpečné pro přerušení a to ty, jejichž název končí `fromISR`. Takové funkce jsou neblokovací a mohou být volány z přerušení.

5 Algoritmus autonomního řízení

Základem tohoto algoritmu je knihovna tfc-k6xf, která zajištuje nastavení hardwaru a časování celého systému. Při inicializaci se nastaví a spustí časovač realizován FlexTimer modulem, neboli FTM. Přerušení tohoto časovače je periodicky spouštěno každou milisekundu a v každé jeho iteraci dojde ke zmenšení proměnných o 1. Tyto proměnné stanovují periodu spuštění odečtu analogových hodnot z kamery a z modulu POLI-TFC. Jakmile dojde k situaci, kdy jedna z těchto proměnných je rovna nule, proměnná je resetována na původní hodnotu periody a je spuštěno přerušení příslušného analogového převodníku, které odečte dané hodnoty a uloží je do paměti.

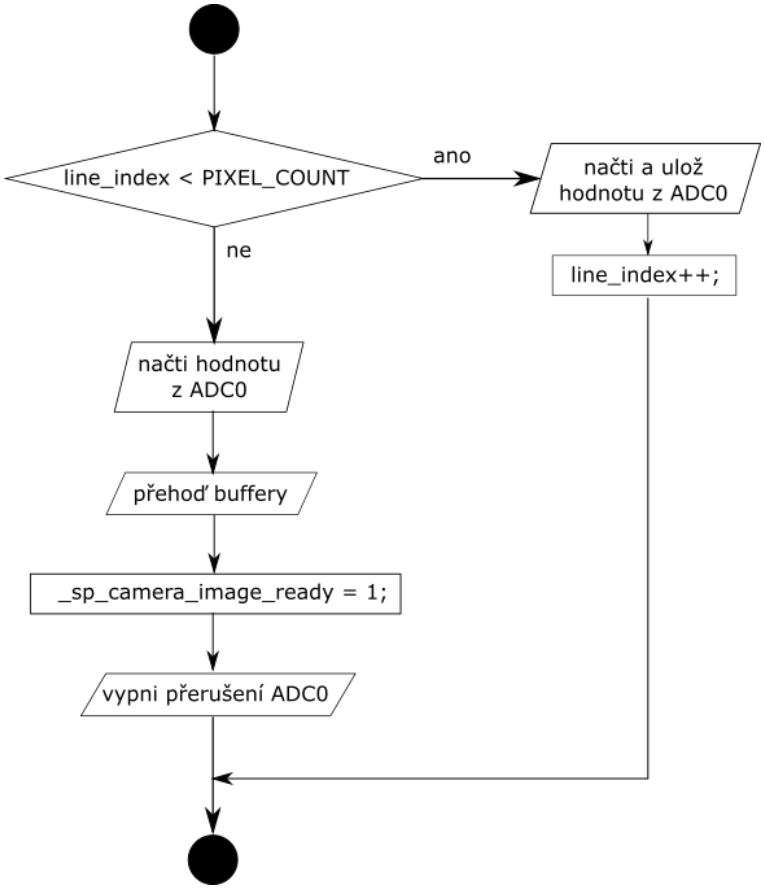


Obrázek 5: Vývojový diagram hlavního časovače

V případě, že dojde k vynulování periody pro záznam z kamery, dojde k nastavení SI pinu, který dá kameře příkaz k uložení obrazu a následně jsou spuštěny hodiny, kterými se bude kamera řídit a vysílat data AD převodníku přes analogový pin. V každé iteraci AD převodníku ADC0 pak dojde k odečtu hodnoty a uložení do dočasného bufferu. Počet iterací tohoto přerušení je roven počtu pixelů kamery, tedy 128. V poslední iteraci dojde k naplnění podmínky, pro

ukončení čtení, buffer, který doposud obsahoval stará data z kamery, bude aktualizován na data nová a příslušný bit proměnné `_s_camera_image_ready` bude nastaven na 1. Tento přístup je zvolen z důvodu dřívějšího použití dvou kamer a aby zůstala tato funkcionalita zachována, kód je ponechán. Jako poslední je volána funkce, pro zakázání přerušení, které tedy až do opětovného povolení časovačem zůstane zakázáno.

Tato akce se opakuje každých 10 milisekund, což je perioda kamery pro časovač a rovněž délka expozice obrazu.



Obrázek 6: Vývojový diagram přerušení ADC0

V hlavním programu je implementován nekonečný cyklus, na jehož počátku je ověřeno, zda je obraz připraven. Jestli obraz připraven není, celý blok cyklu je přeskočen a proces se opakuje. V případě že obraz je připraven, data jsou nakopírována do lokálního bufferu. Tento přístup nazýváme sekvenční a aby systém odpověděl na zachycení snímku kamery v rozumném čase, vytěžuje procesor na maximum. Tento bod je místo, kde je vhodné začít program propojovat s FreeRTOS, neboť zde zajistí synchronizaci a také dostatečnou procesorovou rezervu pro další, méně důležité procesy.

6 Návrh a implementace procesů

V této kapitole je popsán postup zakomponování FreeRTOS do stávající aplikace autonomního řízení modelu auta. Aplikaci je nutné rozdělit do úloh, kterým je přiřazena priorita, čímž díky plánovače docílíme determinismu aplikace. Úlohu autonomního řízení je nutné sesynchronizovat se záznamem z kamery, aby došlo k navázání na ukončení čtení dat z AD převodníku.

6.1 Úloha autonomního řízení modelu

Pro využití funkcionality FreeRTOS, bude zapotřebí upravit knihovnu `tfc-k6xf`. V aktuální podobě není knihovna schopna signalizovat dokončení procesu ukládání obrazu a tím pádem nemůže systém dostatečně rychle zareagovat, aniž by nevytěžoval procesor na 100 %, nebo neměl opožděné reakce. Pro tyto účely FreeRTOS nabízí synchronizaci ve formě semaforů. Nejlepším řešením pro tuto operaci je binární semafor, jenž je schopen jednosměrné signalizace. Aby bylo možné knihovnu i nadále používat i bez FreeRTOS, původní funkcionalita je zachována a je přidáno odevzdání semaforu z přerušení.

Jestliže je odevzdáním semaforu umožněno úloze s vyšší prioritou, než je priorita aktuálně *spuštěné* úlohy, aby byla spuštěna, je nutné zajistit přepnutí kontextu procesoru. Toho lze docílit pomocí parametru `pxHigherPriorityTaskWoken` funkce `xSemaphoreGiveFromISR()`. Jestliže se tento parametr po volání funkce změní na `pdTRUE`, pak je nutné, aby kernel přepnul kontext *spuštěné* úlohy za kontext úlohy s vyšší prioritou. Toto umožňuje funkce `portEND_SWITCHING_ISR()`. Tato operace je znázorněna ve výpisu 1.

```
// Synchronization with higher task woken functionality.
#ifdef SDK_OS_FREE_RTOS
   portBASE_TYPE *xHigherPriorityTaskWoken = pdFALSE;
   xSemaphoreGiveFromISR(Camera_semaphore, xHigherPriorityTaskWoken);
   portEND_SWITCHING_ISR(xHigherPriorityTaskWoken);
#endif
```

Výpis 1: Synchronizace semaforem z přerušení

Po takovéto signalizaci dojde k odblokování úlohy, která čeká na tento semafor. Dalším krokem tedy je vytvoření úlohy, která bude zpracovávat zachycený obraz a ovládat model auta.

Při vytváření úlohy je zapotřebí nejdříve vytvořit statickou funkci, která bude reprezentovat činnost úlohy, stejně tak, jako je tomu u vláken na operačních systémech stolních počítačů.
V této funkci musí být implementován nekonečný cyklus a mělo by být zajištěno, aby úloha nikdy neskončila. V případě, že se úloha vysvobodí z nekonečného cyklu musí být ukončena pomocí funkce `xTaskSuspend()`, nebo `xTaskDelete()`.

Dále je úloze nutné přiřadit velikost statické paměti, kterou je schopna využívat. Pro začátek je tato velikost nastavena na vysokou hodnotu a je tak zajištěn správný chod aplikace. Mikrokontrolery však disponují omezeným množstvím paměti a je třeba mít na mysli, kolik paměti můžeme alokovat. Další důležité nastavení je priorita. Priorita zajistí požadovaný chod aplikace po přidání další úlohy. Prozatím žádnou jinou úlohu než Idle nemáme a prioritu lze tedy nastavit na libovolnou hodnotu od 1, až po `(configMAX_PRIORITIES−1)`.

```
//Autonomous task function declaration
static void Autonomous_task(void *pvParameters);

//Task creation
xTaskCreate(Autonomous_task, "Automatic", 500, NULL, 2, &sd_task_handle);
```

Výpis 2: Vytvoření úlohy autonomního řízení

Implementace funkce úlohy musí, jak již bylo zmíněno, obsahovat nekonečnou smyčku, ve které bude probíhat hlavní proces této úlohy. Předtím, než se úloha dostane do nekonečné smyčky, může definovat a inicializovat proměnné. Jelikož získání bufferu obrazu kamery spočívá v kopírování paměti, je zde vytvořen lokální buffer pro aktuálně zpracovávaný snímek a instance tříd `ProcessedLines`, `CarControl` a `TFC`, čímž se také spustí časovač. V nekonečné smyčce je úloha synchronizována s přerušením, zpracován obraz a uskutečněno řízení modelu.

Synchronizaci s přerušením umožní již vytvořený binární semafor, který se tentokrát úloha pokusí vzít. Funkce `xTaskSemaphoreTake()`, je blokovací funkce, které lze specifikovat timeout. Předáním makra `portMAX_DELAY` této funkci dojde k zablokování úlohy na dobu neurčitou, neboť se úloha nikdy neodblokuje v důsledku vypršení timeoutu. Jakmile přerušení poskytne semafor, úloha se odblokuje a začne zpracovávat data.

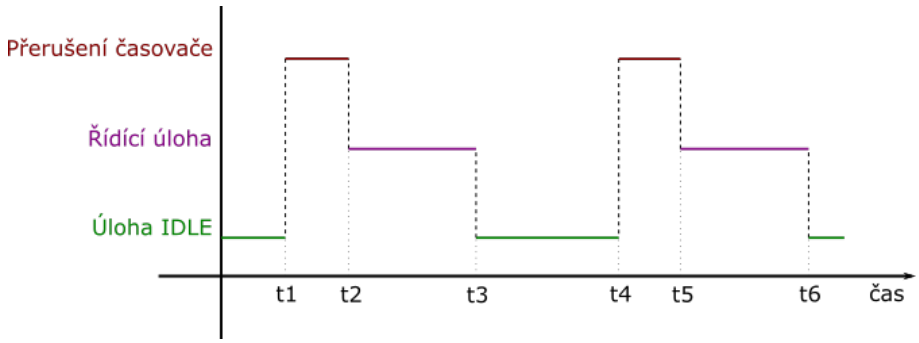
Nejdříve se zkopíruje obraz a to funkcí `memcpy()`. Tato funkce přistupuje do stejné paměti, jako přerušení, které čte data z kamery a proto lze říct, že se jedná o kritickou sekci. V našem případě nemůže nikdy souběh těchto dvou úkonů nastat, neboť je funkce volána ihned po odblokování úlohy. Je však na místě takovéto sekce zabezpečit. FreeRTOS API nabízí funkci `taskENTER_CRITICAL()`, která po dobu kritické sekce zakáže přepínání úloh a také povolování přerušení. V tomto stavu nelze dlouho setrvat, neboť by mohlo dojít ke ztrátě přerušení, která jsou v tomto případě velice důležitá z hlediska časování.

Po nakopírování dojde k ukončení kritické sekce voláním funkce `taskEXIT_CRITICAL()`. Následně pak lze zpracovat již lokální data.

```
// Main body of algorithm task.
for(;;){
   xSemaphoreTake(Camera_semaphore, portMAX_DELAY);
   taskENTER_CRITICAL();
   gpTFC->getImage(lChannel, lpCameraIMG, TFC_CAMERA_LINE_LENGTH);
   taskEXIT_CRITICAL();
   gpLineProcessor->process(lpCameraIMG);
   gpController->control();
}
vTaskSuspend( NULL );
```

Výpis 3: Hlavní program řídící úlohy

Aktuální průběh aplikace, lze vidět na obrázku 7. V čase `t1` dojde k přerušení úlohy Idle přerušením AD převodníku `ADC0`, které je spuštěno hlavním časovačem. K ukončení čtení dat z kamery a odevzdání semaforu přerušením dojde v čase `t2` a ihned po ukončení přerušení se odblokuje řídící úloha, která doposud čekala na semafor. V čase `t3` řídící úloha dokončí zpracování obraz a nastavení řízení a je zablokována při opětovném čekání na semafor. Tímto je spuštěna úloha Idle, neboť žádná jiná úloha zatím neexistuje. Od času `t4` se celý tento proces periodicky opakuje.



Obrázek 7: Aktuální průběh aplikace

V tento moment je základní funkcionalita zajištěna. Aby nebylo autonomní řízení jedinou možností, uživatel bude mít možnost vybrat ze dvou dalších spustitelných úloh. První úloha bude sloužit pro radiově řízený režim modelu auta a druhá pro testovací režim, kterým uživatel otestuje funkčnost jednotlivých hardwarových komponentů systému, například servo motoru.

Oběma těmto úlohám bude přiřazena priorita totožná s prioritou úlohy řídící model auta autonomně, neboť nikdy nebudou spuštěny dvě z těchto úloh současně. Pro umožnění uživateli rozhodovat, která úloha bude spuštěna, přidáme do projektu možnost uživatelského nastavení. Více v kapitole 9.

6.2 Úloha RC řízení modelu

Řízení modelu automobilu pomocí radiových vln se rovněž odvíjí od knihovny tfc-k6xf. K RC řízení modelu je zapotřebí rádiový vysílač a rádiový přijímač s minimálně dvěma kanály. Tyto kanály slouží k ovládání různých komponentů, tedy v tomto případě rychlosti otáček motoru a natočení servo motoru. Radiový přijímač pro každý z těchto kanálů vytváří PWM signál, který má různou střídu v závislosti na přijatém signálu

Tuto střídu detekuje jeden z FTM modulů, který při každé periodě spustí přerušení, které vypočte střídu, uloží délku signálu do proměnné `_hw_tfc_speed_us` a následně signalizuje ukončení výpočtu odevzdáním binárního semaforu, stejně tak, jako tomu bylo u AD převodníku dat z kamery. Tato funkcionalita je inicializována funkcí `HW_TFC_RC_Init()`, která inicializuje hardware a nastaví FTM modul na detekci vzestupných i sestupných hran PWM signálu.

Úloha řídící komponenty bude synchronizována pomocí binárního semaforu, na který bude čekat, dokud nebude dostupný. Timeout tedy bude nastaven na `portMAX_DELAY`. Délku pulsu lze získat pomocí funkce `getRCPulse()`, které je nutné specifikovat kanál, jehož délku pulsu chceme zjistit. Abychom mohli použít délku pulsu pro kontrolu servo motoru a motorů, je nutné tuto hodnotu upravit, aby odpovídala rozsahu pro nastavení jednotlivých komponentů. Tyto úpravy jsou uvedeny ve výpisu 4.

```
// Main body of manual control task
for(;;)
{
   xSemaphoreTake(RC_semaphore, portMAX_DELAY);

   if ((pulse_w = gpTFC->getRCPulse(0)) != 0)
   {
      servo_position = -(pulse_w - 1500)*2;
      gpTFC->setServo_i(channel, servo_position);
   }
   if ((pulse_w = gpTFC->getRCPulse(1)) != 0)
   {
      motor_speed = (pulse_w - 1500);
      gpTFC->setMotorPWM_i(motor_speed, motor_speed);
   }
}
vTaskSuspend( NULL );
```

Výpis 4: Nekonečná smyčka úlohy pro manuální řízení

6.3 Úloha pro test komponentů

V této úloze je využito stejné inicializace, jako při inicializování autonomního ovládání a to funkcí `InitAll()` třídy `TFC`. K testování je zapotřebí ovládacích prvků, bude tedy využito potenciometrů a přepínačů na desce POLI-TFC. Jelikož je komponentů více než ovládacích prvků, fungování úlohy je omezeno vždy pouze na jeden komponent. Tato selekce bude realizována DIP přepínači, takže každý přepínač zapíná a vypíná test jednotlivých komponentů. První přepínač umožní testování motorů, druhý testování servo motoru a čtvrtý přepínač zapne odesílání dat přes ethernetový port.

V případě testování motorů lze kontrolovat rychlost otáčení motorů v obou směrech pomocí dvou potenciometrů. Test serva probíhá pouze pomocí potenciometru označeného POT_1.

7 Vyhodnocení navržené koncepce

7.1 Výpočetní rezervy

V aktuální podobě aplikace obsahuje tři úlohy a to úlohu pro řízení pomocí algoritmu autonomního řízení, úlohu RC řízení a úlohu pro testování komponentů. K měření uplynulého času během zpracování úloh lze využít časovače, integrovaného ve FreeRTOS. Jedná se o softwarový časovač, jehož hodiny lze definovat makrem `configTICK_RATE_HZ`. Jestliže je `configUSE_TIMERS` nastaveno na hodnotu 1, FreeRTOS vytvoří úlohu s maximální možnou prioritou, která bude periodicky spouštěna a ve své iteraci inkrementuje hodnotu tiku.

Z úloh lze voláním funkce `xTaskGetTimerCount()` počet tiků od spuštění vyčíst a na základě frekvence časovače vypočítat čas, který uplynul od posledního čtení.

Tento časovač ale bohužel nemá dostatečné rozlišení času a nelze jej použít pro měření uplynulého času během přerušení. Nastavením frekvence `configTICK_RATE_HZ` na vysokou hodnotu dojde ke zvýšení rozlišení. Podle těchto hodin se však řídí celý kernel FreeRTOS a došlo by tedy k nežádoucím opožděním vlivem častého přerušení kernelu a úlohy softwarového časovače.

K získání přesné hodnoty měření, jak během úloh, tak během přerušení, lze použít hardwarovou periferii `SysTick`. Tato součást čipu ARM Cortex-M4 slouží právě k účelům časování a měření času. Hodnota registru `SYST_RVR` určuje, že bude `SysTick` počítat vždy od této hodnoty směrem dolů. V registru `SYST_CVR` je uložena aktuální hodnota čítače. Výsledkem odečtu novější hodnoty od starší, je počet uskutečněných tiků v průběhu měření. Při 120 MHz hodinách 1 milisekunda trvá 120 000 tiků. `SysTicku` lze také specifikovat, má-li se po dosažení 0 spustit přerušení, nebo ne. Toto slouží k periodickému volání přerušení, odkud se mohou spouštět další součásti systému. Příklad inicializace `SysTick` je vidět ve výpisu 5.[4]

```
// SysTick counting from 0xFFFFFF to 0 without interrupt assertion
SysTick_Type *ST = SysTick;
ST->LOAD |= SysTick_LOAD_RELOAD_Msk;  // Set RELOAD value to 0xFFFFFF
ST->VAL &= ~SysTick_VAL_CURRENT_Msk;  // Clear CURRENT value
ST->CTRL |= SysTick_CTRL_CLKSOURCE_Msk; // Set clock source to processor clock
ST->CTRL &= ~SysTick_CTRL_TICKINT_Msk;  // Do not assert exception on countdown
    to 0
ST->CTRL |= SysTick_CTRL_ENABLE_Msk;  // Set ENABLE bit, to start counting and
   reset value
```

Výpis 5: Příklad inicializace SysTick

Časově kritická je z vytvořených úloh pouze jedna a to úloha autonomního řízení. Měřit se bude čas pouze této úlohy a délky čtení dat z kamery. Naměřené hodnoty jsou k dispozici v tabulce 1.

Tabulka 1: Naměřené časy

Měřený úsek	**čas[ms]**
Záznam obrazu z kamery	2,85
Zpracování dat a řízení modelu	3,68
Celkem	6,53

Z naměřených hodnot je jednoznačně vidět, že při časování 10 milisekund na jeden snímek, ještě 3,57 milisekundy zbývá na další funkcionalitu.

7.2 Využití paměti

Doposud byla úlohám vždy přiřazována vysoká hodnota statické paměti. Velikost této hodnoty závisí na implementaci na daném procesoru. V tomto případě se jedná o 2 bajty a hodnota 500 tedy znamená 1000 bajtů.

FreeRTOS je schopno nejen detekovat přetečení této paměti jednotlivých úloh, ale dokonce také zjistit minimální hodnotu zbývající volné paměti v průběhu spuštění aplikace. Tato hodnota je návratovou hodnotou funkce `uxTaskGetStackHighWaterMark()`. Díky této hodnotě je možné optimalizovat přiřazenou paměť jednotlivých úloh. `INCLUDE_uxTaskGetStackHighWaterMark` musí být nastaveno na hodnotu 1.

Výsledné hodnoty jsou navýšeny o 20 %, aby nedošlo k náhodným přetečením paměti. Hodnoty původní, naměřené a výsledné nalezneme v tabulce 2.

Tabulka 2: Nastavení paměti

	Původní nastavení	**Naměřená rezerva**	**Finální nastavení**
Úloha autonomního řízení	500	141	430
Úloha manuálního řízení	500	376	150
Testovací úloha	500	384	140
Celkem			720

Celkem tedy vytvořené úlohy zabírají 1440 bajtů v SRAM paměti čipu. Tyto hodnoty ale nejsou finální, neboť například kernel, nebo softwarové časovače také vyžadují paměť.

8 Odesílání dat pomocí ethernetového portu

Data, která jsou během jízdy průběžně v systému vypočítána a nastavována, jsou pro uživatele velice hodnotná. Z takovýchto dat lze ověřit správnost algoritmu, či doladit drobné nedostatky, které mohou být jinak těžko zjistitelné. Během jízdy by bylo velice nepohodlné, ba dokonce nemožné, mít řídící desku připojenou kabelem za účelem získávání těchto dat. Jako řešení se nabízí využití přenosu dat pomocí ethernetového portu, kterým řídící desky disponují. Využití pouze ethernetového portu by řešení opět omezilo na použití kabelu a proto je k dispozici miniaturní router, schopný připojit zařízení ke vzdálenému přístupovému bodu. Ze sítě, ke kterému je zařízení připojeno, může libovoné jiné zařízení data přijímat.

8.1 Návrh

Pro realizaci připojení k routeru je použita knihovna lwIP(Lightweight IP)[8], která je dost malá na to, aby mohla být použita v mikrokontrolérech. Nejedná se o úplně nejminimalističtější knihovnu, ale je to knihovna hojně uživaná a tedy i podpora ze strany vývojářů a jiných uživatelů je velice bohatá. Knihovna navíc implementuje funkčnost i pro FreeRTOS, což velice pomůže při implementaci.

Pro odesílání dat je nutné vytvoření struktury, která bude následně ukládána do paketů a odesílána do sítě. Tato struktura bude obsahovat řídící data, tedy dvě struktury knihovny `TFC` a to `tfc_data_s` a `tfc_control_s`. Struktury je nutné zaobalit datovým rámcem zobrazeným v tabulce 3. Data budou odesílána jednosměrně s použitím UDP protokolu.

Tabulka 3: Datový rámec

	STX	**length**	**CMD**	**data**	**ETX**
Počet bajtů	1	2	1	V závislosti na struktuře	1
Hodnota	0x02	0x00 - 0xFF	0x01 / 0x03	Data	0x03

Bajty `STX` a `ETX` označují začátek a konec datového rámce. Tyto bajty jsou využity především pro sériovou komunikaci, kde je nutné začátek a konec dat specifikovat. U zvoleného UDP protokolu tyto bajty až tak nutné nejsou, neboť jsou data uložena v paketu, který je sám o sobě ohraničuje.

Bajt `CMD` slouží k identifikaci obsažené struktury a může v tomto případě nabýt dvou hodnot:

- `CMD_DATA` (0x01) - Označení struktury `tfc_data_s`
- `CMD_CONTROL` (0x03) - Označení struktury `tfc_control_s`

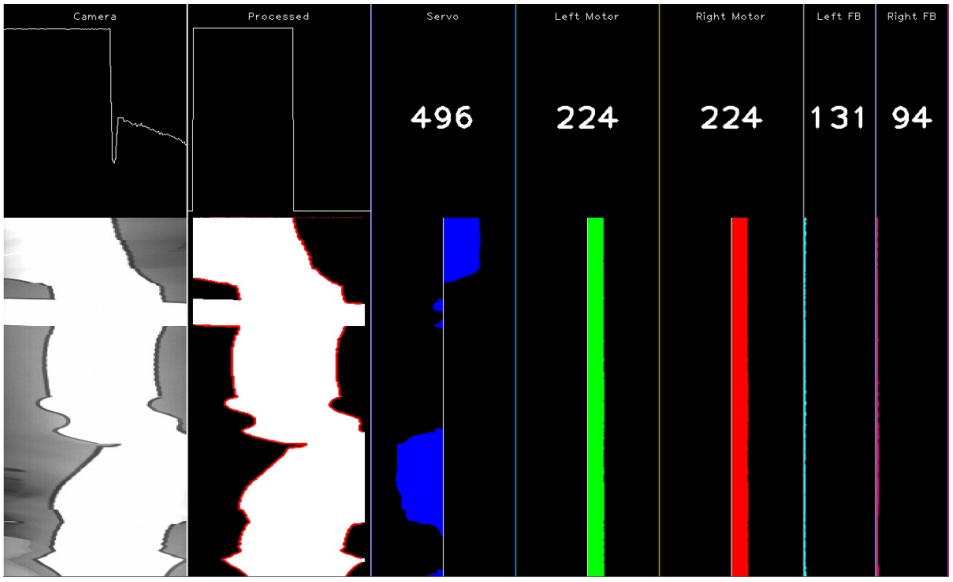
Položka `length` obsahuje velikost struktury v bajtech. Samotná struktura je pak obsažena v položce `data`.

```
// Structure containing analog data and data from camera.
struct tfc_data_s
{
   uint32_t timestamp;      ///< Number of the sample.
   uint16_t adc[ anLast ]; ///< All analog data specified in ::
      tfc_andata_chnl_enum.
   uint8_t dip_sw;          ///< Values of DIP switches.
   uint8_t push_sw;         ///< Values of push buttons.
   uint16_t image[ TFC_CAMERA_LINE_LENGTH ];///< One line from the camera.
   uint32_t _padding;       ///< Padding bytes.
};
// Structure for controlling individual features or receiving their data.
struct tfc_control_s
{
   uint8_t leds;            ///< Values of the 4 LEDs.
   uint8_t  pwm_onoff;      ///< On/off value of the motors {0,1}.
   uint8_t  servo_onoff;    ///< On/off value of the servos {0,1}.
   uint8_t  _padding1;      ///< Padding byte.
   int16_t  pwm[ 2 ];       ///< PWM values of the motors <-::TFC_PWM_MINMAX, ::
      TFC_PWM_MINMAX>.
   int16_t  servo_pos[ 2 ]; ///< Positions of the servos <-::TFC_SERVO_MAX_LR,
      ::TFC_SERVO_MAX_LR>.
};
```

Výpis 6: Struktury ukládané do paketu

Jeden paket pojme hned několik těchto struktur. Aby nebyla síť příliš přetěžována, paket se odešle až když se naplní čtyřmi strukturami. Pakety budou odesílány na broadcast adresu sítě, aby bylo zajištěno, že každé zařízení připojené k síti pakety obdrží.

Aplikace přijímající tyto data je schopna pakety zachytit, vyčíst struktury a pomocí OpenCV data graficky znázornit. Grafické znázornění lze vidět na obrázku 8.



Obrázek 8: Aplikace zobrazující data pořízená během jízdy

8.2 Implementace

Inicializací TCP/IP stacku dojde k vytvoření další úlohy. Tato úloha je definována knihovnou lwIP a má předdefinovanou hodnotu priority 8 a velikost statické paměti 3000. Díky této paměťově náročné úloze je nutné zvětšit heap sektor paměti RAM, o tom však později. Tato úloha zajišťuje inicializaci a veškerou funkcionalitu TCP/IP.

K uskutečnění potřebného odesílání dat je zapotřebí vytvořit ještě jednu úlohu. Aby byl zajištěn plynulý provoz modelu, je této úloze přidělena priorita nižší, než je priorita řízení modelu, tedy 1. V této úloze je nutné vytvořit připojení a inicializovat odesílaný buffer. Pomocí struktury `netconn` se vytvoří připojení, kterému je specifikován protokol, v tomto případě tedy UDP. Dále je vytvořen buffer `netbuf`, který je nainicializován pouze jako reference, to znamená, že není alokován na víc než jeden ukazatel. Následně je připojení `netconn` přidělena IP adresa zařízení, na kterém je lwIP použito a adresa, kam mají být data odesílána, tedy broadcast adresa sítě. Před odesíláním dat chybí ještě poslední krok a tím je reference na odesílaný buffer.

Nyní již mohou být data nahrávána do struktury a strukturu lze odesílat pomocí funkce `netconn_send()`. Veškerá data, která jsou odesílána lze vidět ve výpisu 6. Nastavení, jenž bylo právě provedeno, je vyobrazeno ve výpisu 7.

```
// Structures needed
struct netconn *conn;
struct netbuf *buf;
struct CarData packet[4];

// Establish UDP connection
conn = netconn_new(NETCONN_UDP);
buf = netbuf_new();
netconn_bind(conn, &device_ip, 12344);
netconn_connect(conn, &brcst_ip, 12355);
netbuf_ref(buf, packet, sizeof(packet));
```

Výpis 7: Nastavení úlohy odesílající data

9 Textové uživatelské rozhraní

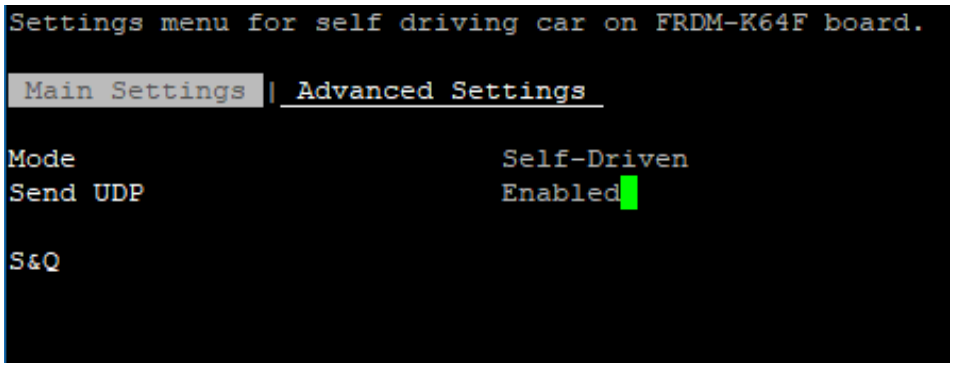
Systém autonomního řízení vozidla obsahuje monoho nastavení, ať už rychlost jízdy, či možnost odesílat data přes Ethernet, které by měl být uživatel schopen nastavit. Aby bylo toto nastavení umožněno a aby bylo nastavení uživatelsky přijatelné, je v aplikaci využito textové uživatelské rozhraní. Vzhledem k tomu, že model automobilu nedisponuje žádným zobrazovacím zařízením, je potřeba data odesílat a přijímat z jiného zařízení pomocí sériové linky UART. Aby však byla eliminována nutnost připojení k jinému zařízení, je potřeba umožnit alespoň minimální nastavení ve formě přepínačů a tlačítek na modulu POLI-TFC.

9.1 Návrh

Uživatelské rozhraní má možnost vykreslení jako BIOS a jako menu. V této práci je použito vykreslení jako menu a to do dvou záložek.

První záložka vyobrazená na obrázku 9 s názvem "Main Settings"obsahuje 3 položky:

- `Mode` - Mód, ve kterém se systém má spustit, tedy Autonomní, Manuální, nebo Testovací.
- `Send UDP` - Proměnná, která určuje, zdali bude nainicializován TCPIP stack s aplikací odesílající data přes Ethernet.
- `S&Q` - Funkce, která ukončí menu, a spustí vybranou úlohu s daným nastavením.



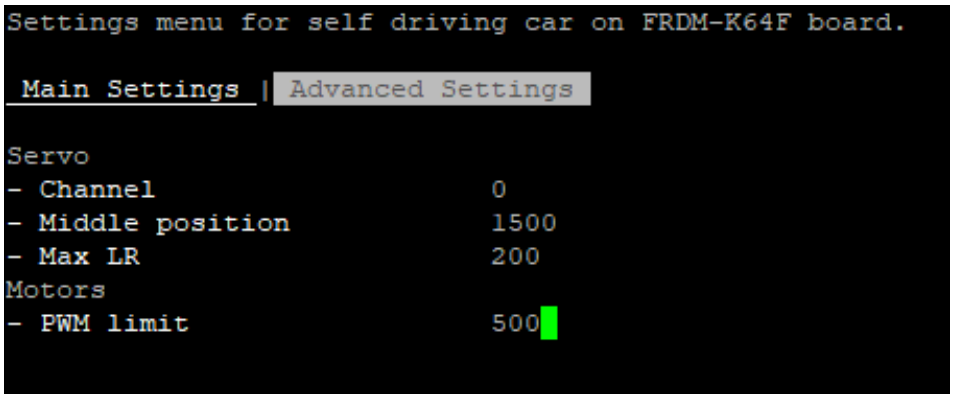
Obrázek 9: Záložka Main Settings

Druhá záložka vyobrazená na obrázku 10 s názvem "Advanced Settings"obsahuje tyto položky:

1. Servo
    - `Channel` - Kanál modulu POLI-TFC, do kterého je servo připojeno.
    - `Middle position` - Hodnota, určující středovou pozici natočení kol.
    - `Max LR` - Maximální odchylka od středové hodnoty.

2. Motors

   - `PWM limit` - Limit pro hodnotu PWM signálu H mostů.



Obrázek 10: Záložka Advanced Settings

K eliminaci nutnosti užití dalšího zařízení pro nastavení, respektive pro výběr módů, je v aplikaci možnost ukončení nastavení pomocí vstupů modulu POLI-TFC. V případě užití tohoto ukončení nastavení dojde k nastavení hodnot, stanovených knihovnou `TFC`.

K ukončení a nastavení módu `TEST` stačí stisknout tlačítko `B`. Pro rozhodování, mezi módem `AUTO` a `MANUAL` slouží přepínač na pozici 1. Je-li zapnutý, stisknutí tlačítka `A` spustí radiově řízený mód a tedy úlohu RC řízení. Je-li však vypnutý, aplikace se po stisknutí tlačítka `A` spustí v autonomním režimu.

Pro signalizaci aktuálně spuštěného režimu je využita RGB dioda, která je součástí obou použitých vývojových kitů. Jednotlivým režimům jsou přiřazeny tyto barvy:

- `Manuální režim` - Červená (#FF0000)
- `Autonomní režim` - Zelená (#00FF00)
- `Testovací režim` - Modrá (#0000FF)

9.2 Implementace

Aby mohla být knihovna sprostředkující textové uživatelské rozhraní použita, je nutné nejdříve implementovat její funkce, které se liší v závislosti na procesoru a použitém operačním systému. Jedná se o funkce čtení a zápisu sériové linky, převody mezi čísly a charaktery, nastavení zpoždění a aktualizace čítačů, použitých pro zjištění počtu přijatých charakterů ze sériové linky.

Další krok je vytvoření menu a to definováním struktur `ucMenu`, `ucMenuPages` a `ucMenuItem`.

Struktura `ucMenu` popisuje základní vlastnosti menu jako je titulní název menu, rozměry a řádkování menu a v neposlední řadě obsahuje funkce, které byly naimplementovány v předchozím kroku. Struktura `ucMenuPages` obsahuje záložky menu a tedy instance struktury ucMenuItem, která obsahuje jednotlivé řádky dané záložky s popisem a referencemi na čtecí a zapisující

funkce daného řádku. V těchto funkcích lze naimplementovat kontroly vstupu pro případ, že by uživatel zadal chybný vstup.

Pro účely sériové komunikace existují v API funkce jak pro inicializaci pinů, tak pro příjem a odesílání dat. Funkce `UART_RTOS_Receive()` je blokovací funkce a tedy zablokuje úlohu, která se pokusí číst ze sériové linky, když zrovna žádná data nepřicházejí. Z tohoto důvodu je v této práci použito dvou úloh pro zprostředkování této komunikace a to úlohy pro čtení ze sériové linky a úlohy pro zprostředkování menu a tedy i zápis do sériové linky.

Úloha, která čte data ze sériové linky má vyšší prioritu, než ta, která do ní zapisuje. Tímto je zajištěna responzivita systému, takže nemůže dojít ke značnému zpoždění mezi stisnutím klávesy a zobrazením akce v menu. Přijatý znak je dále vložen do fronty, která je pro tyto účely vytvořena a inicializována na velikost 16 znaků, odkud je znak následně vyčten úlohou zprostředkující menu.

Důležitá je funkce ukončení menu. V této funkci je knihovně `tfc` nastaveno vybrané nastavení a umožní se zde spuštění jedné z úloh a vymazání úloh zbylých. Pro ulehčení spouštění jednotlivých úloh jsou všechny úlohy již vytvořené suspendovány, kromě úloh souvisejících s menu. Tohoto lze dosáhnout funkcí `vTaskSuspend()` volanou na počátku každé vybrané úlohy.

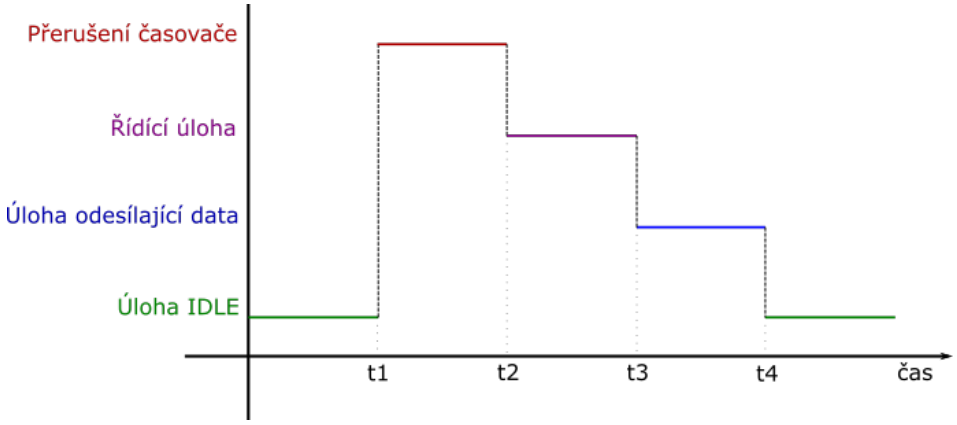
Je-li zvolena možnost odesílání dat přes Ethernet, vytvoří se úlohy spojené s touto funkcionalitou. Dle volby režimu se inicializují potřebné periferie a smažou se nepotřebné úlohy. Následuje odsuspendování zvolené úlohy. Je nežádoucí, aby odsuspendovaná úloha ihned přerušila úlohy menu. Proto je úlohám menu přiřazena vyšší priorita, než je priorita spouštěných úloh. Tímto je zajištěno, že úloha je sice připravena, ale nespustí se dokud nebude ukončena úloha menu. Získaný výpočetní čas je využit ke smazaní čtecí úlohy menu a následně i menu samotného.

V tento moment úloha Idle vyčistí paměť po vymazaných úlohách a dojde ke spuštění vybraného režimu.

10 Vyhodnocení finální aplikace

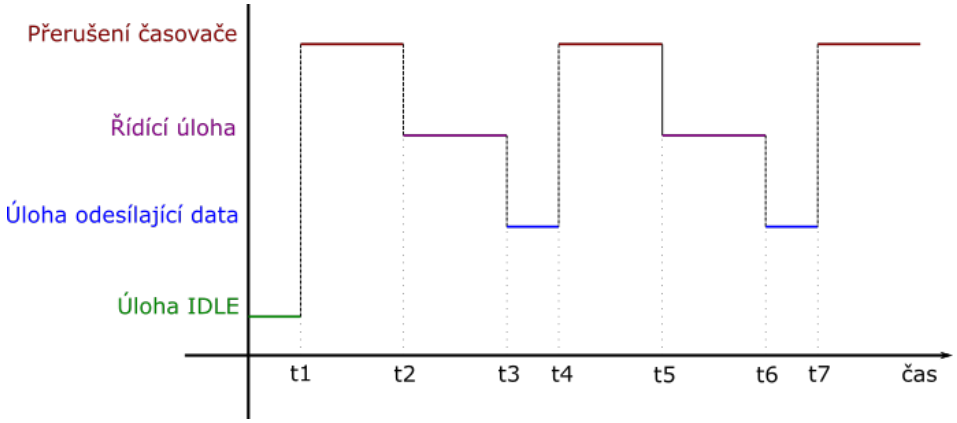
Přidáním dvou nových funkcionalit jsem docílil systému, který je uživatelsky nastavitelný a schopný odesílání dat během jízdy. Textové uživatelské rozhraní nijak neomezuje kritické časování systému, neboť je spuštěno ještě před spuštěním časovače. Na druhou stranu odesílání dat v průběhu jízdy vyžaduje výpočetní čas, který je v tomto případě omezen.

V naprosto ideálním případě by čas od odblokování úlohy odesílající data po její zablokování nebyl delší, než časová rezerva, která nám zbyla po prvním vyhodnocení aplikace v kapitole 7, tedy 3,47 milisekundy. V takovém případě by ještě před spuštěním odečtu dat z kamery došlo ke špuštění úlohy Idle. Takovýto průběh je vidět na obrázku 11.



Obrázek 11: Ideální časování

Naopak, kdyby doba potřebná pro exekuci úlohy byla delší, než stanovený limit, docházelo by k přeskakování odesílaných dat. Aplikace by pořád splňovala potřebný efekt řízení vozu, avšak uživatel by nebyl schopen pozorovat chování algoritmu s každým snímkem, neboť by s největší pravděpodobností docházelo k opakovanému přeskakování snímků. Graf takovéhoto časování můžeme vidět na obrázku 12.



Obrázek 12: Akceptovatelné časování

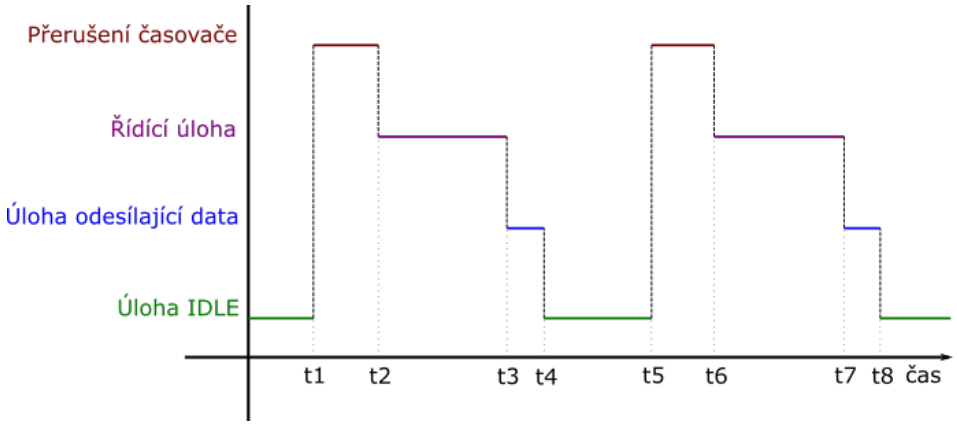
Pro určení výpočetní a paměťové náročnosti byla použita stejná metodika, jako v kapitole 7.

Tabulka 4: Naměřené časy

Měřený úsek	**čas[ms]**
Záznam obrazu z kamery	2,84
Zpracování dat a řízení modelu	3,71
UDP bez odeslání	0,04
UDP s odesláním	0,36
Celkem	6,95

Z naměřených časů, potřebných pro zpracování jednotlivými částmi systému, jde jednoznačně vidět, že přidání úlohy pro odesílání dat přes ethernetový port nijak neomezuje dosavadní časování. V aplikaci tak je stále dostatek času pro další funkcionalitu. Bylo tedy dosaženo časování podobného tomu ideálnímu na obrázku 11.

Výsledný průběh aplikace je vyobrazen na obrázku 13, kde v čase `t1` dojde k přerušení úlohy Idle přerušením AD převodníku `ADC0`. Po ukončení čtení dat z kamery v čase `t2` přerušení odevzdá semafor, který odblokuje úlohu automatického řízení modelu. V čase `t3` úloha dokončí svou činnost předáním semaforu úloze odesílající data přez UDP. V čase `t4` pak dojde k zablokování úlohy odesílající data a spuštena je opět úloha Idle. Celý proces se od času `t5` opakuje.



Obrázek 13: Výsledné časování

Při testovací jízdě model automobilu reagoval jednoznačně dost rychle na jednotlivé situace a soudím tedy, že změna časování není nutná. V případě, že by aplikace nebyla schopna reagovat dostatečně rychle na tyto situace, tak je možné zkrátit periodu čtení dat z kamery. Časová rezerva je po přídání všech funkcionalit stále 3,05 milisekund a tedy časování by mohlo být zkráceno na periodu 8 milisekund. Tento krok však není nutný.

Provedl jsem rovněž testy obou vedlejších režimů, tedy režimu RC řízení modelu a testovacího režimu. Oba tyto režimy splňovaly zamýšlenou funkcionalitu bez jakýchkoliv potíží. Ovládání pomocí RC bylo responzivní a v testovacím režimu jsem úspěšně ověřil funkčnost veškerých testovaných komponentů.

Tabulka 5: Nastavení paměti

	Původní nastavení	**Naměřená rezerva**	**Finální nastavení**
Úloha TCPIP	3000	1832	1400
Úloha odesílající data	1000	262	900
Úloha čtení ze sériové linky	500	372	150
Úloha TUI	500	331	200
Celkem			2650

V tabulce 5 jsou hodnoty původní, naměřené a nastavené paměti nově přidaných úloh. Hodnoty původních úloh zůstaly nezměněny a lze je vidět v tabulce 2.

Jak již bylo zmíněno dříve, úlohy spojené s knihovnou `lwIP` jsou velice náročné na paměť a heap sekce SRAM paměti pro FreeRTOS musela být zvětšena z původních 10240 na 20480. Tuto hodnotu lze nastavit v makru `configTOTAL_HEAP_SIZE` v hlavičkovém souboru `FreeRTOSConfig.h`.

Program po kompilaci zabírá 154728 bajtů a tedy 14.76 % celkové paměti Flash a 85608 bajtů v SRAM paměti, což je rovno 43.54 %. Mikropočítač má tedy stále velké rezervy.

Hodnoty pro desku FRDM-K66F nebyly zjištěny z důvodu nefunkčnosti programu na této desce. Aplikace sice fungovala, ale časování této desky nebylo v pořádku a bude předmětem dalšího testování aplikace na této desce.

11 Závěr

Cílem této práce bylo analyzovat stávající program autonomního řízení modelu auta a převést jej do FreeRTOS. Rozdělit dosavadní funkcionalitu na úlohy a zajistit synchronizaci těchto úloh s hardwarovým vybavením. Dále pak doplnit aplikaci o záznam a odesílání dat, stejně tak, jako o příjem řídících a konfiguračních dat. Výslednou koncepci otestovat a vyhonotit.

Výsledný program je rozdělen do sedmi úloh. Přiřazením patřičných priorit těmto úlohám bylo docíleno determinismu, který vedl k úspěšnému spuštění a otestování programu na desce FRDM-K64F.

Do aplikace byla úspěšně přidána možnost odesílat data přez ethernetový port. Rovněž se podařilo aplikaci obohatit o příjem konfiguračních a řídících dat pomocí sériové linky a textového uživatelského rozhraní.

Nad rámec zadání byla aplikace rozšířena o použití na desce FRDM-K66F s výkonnějším procesorem. Úspěšného spuštění bohužel nebylo dosaženo z důvodů chybného časování. Nastavení časování pro tuto desku bude předmětem dalšího zkoumání a testování programu.

Testováním bylo zjištěno, že program plně vyhovuje konkrétnímu zadání, tedy jízdě po trati. Textové uživatelské rozhraní také plně splňuje svůj účel a odesílání dat při testu v laboratoři probíhalo se 100% úspěšností zachycení odesílaných dat. Z hlediska časování bylo dosaženo ideálních výsledků a to dokonce s velkou časovou rezervou, která může být využita pro další funkcionalitu programu.

Literatura

[1] IHN, Vojtěch. Autonomní řízení auta - optimalizace detekce dráhy [online]. Ostrava, 2018 [cit. 2019-04-27]. Dostupné z: http://hdl.handle.net/10084/128642. Bakalářská práce. Vysoká škola báňská - Technická univerzita Ostrava.

[2] HANSLÍK, Filip. Autonomní řízení auta - optimalizace řízení rychlosti [online]. Ostrava, 2018 [cit. 2019-04-27]. Dostupné z: http://hdl.handle.net/10084/128641. Bakalářská práce. Vysoká škola báňská - Technická univerzita Ostrava.

[3] GREGOROVIČ, Peter. Uživatelské rozhraní pro mikropočítač [online]. Ostrava, 2015 [cit. 2019-04-27]. Dostupné z: http://hdl.handle.net/10084/108900. Bakalářská práce. Vysoká škola báňská - Technická univerzita Ostrava.

[4] K64 Sub-Family Reference Manual [online]. [cit. 2019-04-27]. Dostupné z https://www.nxp.com/files-static/microcontrollers/doc/ref_manual/K64P144M120SF5RM.pdf

[5] MCUXpresso SDK API Reference Manual [online]. [cit. 2019-04-27]. Dostupné z https://mcuxpresso.nxp.com/api_doc/dev/116/modules.html

[6] The FreeRTOS Kernel [online]. [cit. 2019-04-27]. Dostupné z https://freertos.org/

[7] The FreeRTOS Reference Manual [online]. [cit. 2019-04-27]. Dostupné z https://freertos.org/Documentation/FreeRTOS_Reference_Manual_V10.0.0.pdf

[8] Lightweight IP stack API [online]. [cit. 2019-04-27]. Dostupné z http://www.nongnu.org/lwip/2_0_x/raw_api.html

[9] FRDM-K64F: Freedom Development Platform for Kinetis K64, K63 and K24 MCUs [online]. [cit. 2019-04-27]. Dostupné z https://www.nxp.com/support/developer-resources/evaluation-and-development-boards/freedom-development-boards/mcu-boards/freedom-development-platform-for-kinetis-k64-k63-and-k24-mcus:FRDM-K64F

[10] FRDM-K66F: Freedom Development Platform for Kinetis K66, K65 and K26 MCUs [online]. [cit. 2019-04-27]. Dostupné z https://www.nxp.com/support/developer-resources/evaluation-and-development-boards/freedom-development-boards/mcu-boards/freedom-development-platform-for-kinetis-k66-k65-and-k26-mcus:FRDM-K66F

[11] Freescale Cup Shield for the Freedom KL25Z [online]. [cit. 2019-04-27]. Dostupné z https://community.nxp.com/docs/DOC-93914

[12] The NXP Cup EMEA [online]. [cit. 2018-04-24]. Dostupné z https://community.nxp.com/groups/tfc-emea

[13] LANDZO TFC Car model kit [online]. [cit. 2019-04-27]. Dostupné z http://www.landzo.com/index.php?route=product/product&product_id=95

A Soubor tfc-k6xf.h

```
#ifndef ___TFC_K6xF_H
#define ___TFC_K6xF_H

/** @file
  tfc_k64f.h
*/

#define NL "\r\n"

#ifdef SDK_OS_FREE_RTOS

#include "FreeRTOS.h"
#include "queue.h"
#include "semphr.h"
#include "task.h"

extern SemaphoreHandle_t RC_semaphore;
extern SemaphoreHandle_t Camera_semaphore;

#endif

/**
@name Motors
@{
*/

/**
 * @brief    Initialization  of  both H-Bridges for motors.
 *
 * This function  initializes  two pairs  of PWM channels.
 * The pins A1/A2 for the bridge/motor A and pins B1/B2 for the bridge/motor B are used on the board.
 * One pin is also  initialized  to enable/disable  both H-bridges.
 */

void HW_TFC_MOTOR_Init();

/**
 * @brief    Switch On or Off motors.
 *
 * This function enables or  disables  PWM of both motors.
 * When onoff is set  to  non-zero, function sets  PWM of both motors to 0, starts FTM timer and both H-
      bridges are enabled.
 * When onoff is set  to  zero,  function  sets  PWM of both motors to 0, stops FTM timer and both H-bridges
      are disabled.
 * Calling  this  function  with the same value as the current  status  causes the  call  to  be ignored.
 *
```

```
 * @param onoff The Non-zero value - ON, the 0 value - OFF.
 */
void HW_TFC_MOTOR_OnOff( int onoff );

/**
 * @brief    Set PWM power of motors A and B in range <-::TFC_PWM_MINMAX, ::
      TFC_PWM_MINMAX>.
 *
 * This function takes two values as arguments, where each belongs to one motor.
 * If the value given is not within the allowed range, it is limited into range <-::TFC_PWM_MINMAX,
      ::TFC_PWM_MINMAX>.
 * (For the safety reasons both values are limited into range <-::HW_TFC_PWM_MAX, ::
      HW_TFC_PWM_MAX>
 * Selected value is used to calculate corresponding length of a PWM time period for each channel.
 * PWM values are then set to the both pairs of channels of the timer.
 *
 * @param motor_a  Value for motor A.
 * @param motor_b Value for motor B.
 */
void HW_TFC_MOTOR_SetPWM( int motor_a, int motor_b );

/**
* @}
*/

/**
@name Servos
@{
*/

/**
 * @brief     Initialization of servos.
 *
 * This function initializes FTM timer and allows updating by a Software Trigger.
 * The pins SERVO 0 and 1 are used on the board.
 */
void HW_TFC_SERVO_Init();

/**
 * @brief    Switch On or Off servos.
 *
 * This function enables or disables FTM timer of both servos.
 * When onoff is set to non-zero value, the function enables FTM timer.
 * When onoff is set to 0, function enables interrupt, which disables the timer at the end of the current
      PWM period,
 * to prevent a distortion of the current pulse.
 * Calling this function with the same value as the current status causes the call to be ignored.
 //
```

```
 *
 * @param onoff  The non-zero value - ON, the 0 value - OFF
 */
void HW_TFC_SERVO_OnOff( int onoff );

/**
 * @brief Set position of servo in microseconds.
 *
 * This function takes two arguments, first is used to determine which servo is being set.
 * Second parameter is value to set the position of the servo.
 * If the value given is not within the allowed range, it is set into range
 * <-::TFC_SERVO_MAX_LR+::TFC_SERVO_DEFAULT_CENTER, ::TFC_SERVO_MAX_LR+::
     TFC_SERVO_DEFAULT_CENTER>.
 * Selected value is used to calculate corresponding length of a PWM time period.
 * PWM value is then assigned to selected servo's FTM timer channel.
 *
 * @param servo The servo channel 0 or 1
 * @param pos  Position of servo in microseconds.
 */
void HW_TFC_SERVO_Set( int servo, int pos );

/**
* @}
*/

/**
@name AD converter
@{
*/

/**
 * @brief    Initialization of AD converter for analog inputs.
 *
 * This function initializes ADC, which reads values of potentiometers, H-bridge feedbacks and battery
     voltage.
 */
void HW_TFC_ANDATA_Init();

/**
 * @brief   Switch On or Off the AD conversion.
 *
 * This function enables or disables interrupt, which starts the ADC and converts all values.
 * Calling this function with the same value as the current status causes the call to be ignored.
 *
 * @param onoff The non-zero value - ON, the 0 value - OFF
 */
void HW_TFC_ANDATA_OnOff( int onoff );
```

```
/**
 * @brief    Get current value of selected analog channel.
 *
 * @param chnl Selected channel.
 * @return      Value of the analog channel.
 */
uint16_t HW_TFC_ANDATA_getVal( uint32_t chnl );

/**
* @}
*/

/**
@name Cameras
@{
*/

/**
 * @brief     Initialization   of the camera interface.
 *
 * This function   initializes   ADC for cameras.
 * The pins CAMERA 0 and 1 are used on the board.
 */
void HW_TFC_CAMERA_Init();

/**
 * @brief Switch cameras On or Off
 *
 * This function enables or disables interrupt, which starts the ADC conversion of pixels [::
      TFC_CAMERA_LINE_LENGTH] for both cameras.
 * Calling this function with the same value as the current status causes the call to be ignored.
 *
 * @param onoff The non-zero value - ON, the 0 value - OFF
 */
void HW_TFC_CAMERA_OnOff( int onoff );

/**
 * @brief Function to check whether the image is processed or not.

 * @return 0 - not ready, everything else - ready
 */
uint32_t HW_TFC_CAMERA_isReady();

/**
 * @brief Function to copy camera image of selected camera from internal buffer into array.
 *
 * @param chnl A camera channel.
 * @param line Pointer to an array, where the picture should be stored.
```

```
 * @param len Size of an array, where the image is being loaded.
 */
void HW_TFC_CAMERA_getImage( int chnl, uint16_t *line, int len );

/**
* @}
*/
///Number of up/down counters.
#define HW_TFC_NUM_TICKERS    4
///Timer counters, counting up.
extern volatile uint32_t HW_TFC_Ticker_Up[HW_TFC_NUM_TICKERS];
///Timer counters, counting down.
extern volatile uint32_t HW_TFC_Ticker_Down[HW_TFC_NUM_TICKERS];
///Number of captured camera images.
extern volatile uint32_t HW_TFC_TimeStamp;


/**
@name Timer
@{
*/

/**
 * @brief Initialization of timer.
 *
 * This function initializes FTM timer, which ensures fluent run of the program.
 * The camera and analog values ADCs are being enabled periodically.
 */
void HW_TFC_Ticker_Init();

/**
 * @brief Switch On or Off the timer.
 *
 * @param onoff The non-zero value - ON, the 0 value - OFF
 */
void HW_TFC_Ticker_OnOff( int onoff );

/**
* @}
*/

/**
@name Ports & Pins
@{
*/

/**
 * @brief Initialization of PORTs and Pins.
```

```
 */
void HW_TFC_IOPIN_Init();

/**
 * @brief Get state of DIP switches.
 * @return Value where last four bits represents state of DIP switches.
 */
uint32_t HW_TFC_INPIN_DIPSwitch();

/**
 * @brief Get state of push buttons.
 * @return Value where last two bits represents the state of push switches.
 */
uint32_t HW_TFC_INPIN_ABSwitch();

/**
 * @brief Settings of LEDs 1-4
 * @param leds Low nibble specify state of four LEDs
 */
void HW_TFC_OUTPIN_LEDs( uint32_t leds );

/**
* @brief Settings of individual LEDs.
* @param led LED ID <0,3>
* @param level Logical 1 - ON, logical 0 - OFF
*/

void HW_TFC_OUTPIN_LED( uint32_t led, uint32_t level );

/**
* @}
*/

/**
@name RC Speed pins
@{
*/

/**
* @brief  Initialization  of SPEED pins and FTM timer.

* The pins SPEED 0 and 1 are used on the board.
*/
void HW_TFC_RC_Init();

/**
* @brief Function to turn RC control on or off.
*
```

```
* This function enables/disables FTM timer and interrupts.
* Calling this function with the same value as the current status causes the call to be ignored.
*
*  @param onoff The non-zero value - ON, the 0 value - OFF
*/
void HW_TFC_RC_OnOff( uint32_t onoff );

/**
* @brief Function to get the width of pulse.
* @param channel Channel of SPEED pins
* @return Width of pulse in microseconds.
*/
uint32_t HW_TFC_RC_getPulse_us( uint32_t channel );

/**
* @}
*/

#endif //TFC_K6xF_H
```

Výpis 8: tfc-k6xf.h

B Soubor FreeRTOSConfig.h

```
/*
 * FreeRTOS Kernel V10.0.1
 * Copyright (C) 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this software and associated documentation files (the "Software"), to deal in
 * the Software without restriction, including without limitation the rights to
 * use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
 * the Software, and to permit persons to whom the Software is furnished to do so,
 * subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in all
 * copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
     FITNESS
 * FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS
      OR
 * COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
     WHETHER
 * IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
 * CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 *
 * http://aws.amazon.com/freertos
 * http://www.FreeRTOS.org
 */

#ifndef FREERTOS_CONFIG_H
#define FREERTOS_CONFIG_H

/*-----------------------------------------------------------
 * Application specific definitions.
 *
 * These definitions should be adjusted for your particular hardware and
 * application requirements.
 *
 * THESE PARAMETERS ARE DESCRIBED WITHIN THE 'CONFIGURATION' SECTION OF THE
 * FreeRTOS API DOCUMENTATION AVAILABLE ON THE FreeRTOS.org WEB SITE.
 *
 * See http://www.freertos.org/a00110.html.
 *----------------------------------------------------------*/

#define configUSE_PREEMPTION            1
#define configUSE_TICKLESS_IDLE         0
#define configCPU_CLOCK_HZ              (SystemCoreClock)
```

```
#define configTICK_RATE_HZ                    ((TickType_t)200)
#define configMAX_PRIORITIES                  10
#define configMINIMAL_STACK_SIZE              ((unsigned short)90)
#define configMAX_TASK_NAME_LEN               20
#define configUSE_16_BIT_TICKS                0
#define configIDLE_SHOULD_YIELD               1
#define configUSE_TASK_NOTIFICATIONS          1
#define configUSE_MUTEXES                     1
#define configUSE_RECURSIVE_MUTEXES           1
#define configUSE_COUNTING_SEMAPHORES         1
#define configUSE_ALTERNATIVE_API             0 /* Deprecated! */
#define configQUEUE_REGISTRY_SIZE             8
#define configUSE_QUEUE_SETS                  0
#define configUSE_TIME_SLICING                0
#define configUSE_NEWLIB_REENTRANT            0
#define configENABLE_BACKWARD_COMPATIBILITY   1
#define configNUM_THREAD_LOCAL_STORAGE_POINTERS 5
#define configUSE_APPLICATION_TASK_TAG        0

/* Memory allocation related definitions. */
#define configSUPPORT_STATIC_ALLOCATION       0
#define configSUPPORT_DYNAMIC_ALLOCATION      1
#define configTOTAL_HEAP_SIZE                 ((size_t)(20480))
#define configAPPLICATION_ALLOCATED_HEAP      0

/* Hook function related definitions. */
#define configUSE_IDLE_HOOK                   0
#define configUSE_TICK_HOOK                   0
#define configCHECK_FOR_STACK_OVERFLOW        0
#define configUSE_MALLOC_FAILED_HOOK          0
#define configUSE_DAEMON_TASK_STARTUP_HOOK    0

/* Run time and task stats gathering related definitions. */
#define configGENERATE_RUN_TIME_STATS         0
#define configUSE_TRACE_FACILITY              1
#define configUSE_STATS_FORMATTING_FUNCTIONS  0

/* Co-routine related definitions. */
#define configUSE_CO_ROUTINES                 0
#define configMAX_CO_ROUTINE_PRIORITIES       2

/* Software timer related definitions. */
#define configUSE_TIMERS                      1
#define configTIMER_TASK_PRIORITY             (configMAX_PRIORITIES - 1)
#define configTIMER_QUEUE_LENGTH              10
#define configTIMER_TASK_STACK_DEPTH          (configMINIMAL_STACK_SIZE * 2)

/* Define to trap errors during development. */
```

```
#define configASSERT(x) if((x) == 0) {taskDISABLE_INTERRUPTS(); for (;;);}

/* Optional functions - most linkers will remove unused functions anyway. */
#define INCLUDE_vTaskPrioritySet            1
#define INCLUDE_uxTaskPriorityGet           1
#define INCLUDE_vTaskDelete                 1
#define INCLUDE_vTaskSuspend                1
#define INCLUDE_vTaskDelayUntil             1
#define INCLUDE_vTaskDelay                  1
#define INCLUDE_xTaskGetSchedulerState      1
#define INCLUDE_xTaskGetCurrentTaskHandle 1
#define INCLUDE_uxTaskGetStackHighWaterMark 0
#define INCLUDE_xTaskGetIdleTaskHandle      0
#define INCLUDE_eTaskGetState               0
#define INCLUDE_xTimerPendFunctionCall      1
#define INCLUDE_xTaskAbortDelay             0
#define INCLUDE_xTaskGetHandle              0
#define INCLUDE_xTaskResumeFromISR          1

#ifdef __NVIC_PRIO_BITS
/* __BVIC_PRIO_BITS will be specified when CMSIS is being used. */
#define configPRIO_BITS __NVIC_PRIO_BITS
#else
#define configPRIO_BITS 4 /* 15 priority levels */
#endif

/* The lowest interrupt priority that can be used in a call to a "set priority"
function. */
#define configLIBRARY_LOWEST_INTERRUPT_PRIORITY ((1U << (configPRIO_BITS)) - 1)

/* The highest interrupt priority that can be used by any interrupt service
routine that makes calls to interrupt safe FreeRTOS API functions. DO NOT CALL
INTERRUPT SAFE FREERTOS API FUNCTIONS FROM ANY INTERRUPT THAT HAS A HIGHER
PRIORITY THAN THIS! (higher priorities are lower numeric values. */
#define configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY 2

/* Interrupt priorities used by the kernel port layer itself. These are generic
to all Cortex-M ports, and do not rely on any particular library functions. */
#define configKERNEL_INTERRUPT_PRIORITY (configLIBRARY_LOWEST_INTERRUPT_PRIORITY
    << (8 - configPRIO_BITS))
/* !!!! configMAX_SYSCALL_INTERRUPT_PRIORITY must not be set to zero !!!!
See http://www.FreeRTOS.org/RTOS-Cortex-M3-M4.html. */
#define configMAX_SYSCALL_INTERRUPT_PRIORITY (
    configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY << (8 - configPRIO_BITS))

/* Definitions that map the FreeRTOS port interrupt handlers to their CMSIS
standard names. */
#define vPortSVCHandler SVC_Handler
```

```
#define xPortPendSVHandler PendSV_Handler
#define xPortSysTickHandler SysTick_Handler

#endif /* FREERTOS_CONFIG_H */
```

Výpis 9: FreeRTOSConfig.h

C Obsah elektronické přílohy

```
attachment
├── doc ... Stažená dokumentace.
│   ├── FreeRTOS_Reference_Manual_V10.0.0.pdf
│   └── K64_reference_manual.pdf
├── thesis
│   ├── Thesis_Latex_source
│   │   ... Zdrojové kódy této práce pro LaTeX.
│   └── thesis.pdf
│       ... Digitální forma této práce.
└── sources
    ├── auto_alg
    │   ... Zdrojové kódy pro zpracování obrazu a řízení modelu.
    ├── lwIP
    │   ... Zdrojové kódy knihovny lwIP a aplikace odesílající data
    │       přez ethernetový port.
    ├── tfc-k6xf
    │   ... Zdrojové kódy knihovny tfc-k6xf.
    ├── TUI
    │   ... Zdrojové kódy textového uživatelského rozhraní.
    ├── FreeRTOSConfig.h
    │   ... Konfigurační soubor FreeRTOS.
    ├── FRTOS_K64F.cpp
    │   ... Hlavní program aplikace.
    └── GlobalDefinitions.h
        ... Soubor obsahující globální definice.
```